TSBB15
Computer Vision

Lecture 6
Clustering and Learning

Per-Erik Forssén

Today's topics

- **Machine learning in Computer Vision**
  - Use cases
- **Main algorithms**
  - K-means clustering
  - Mixture models and EM
  - Background models
- **Related algorithms**
  - Meanshift clustering
  - Generalised Hough Transforms (GHT)
  - Channel clustering

Machine Learning in Computer Vision

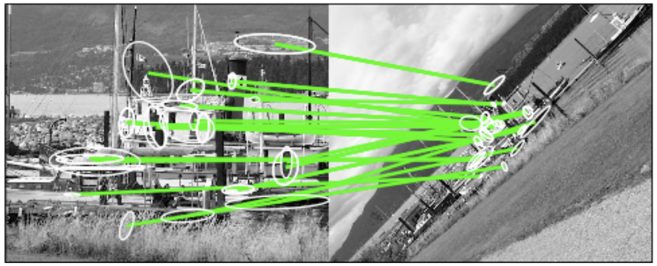
- Machine Learning(ML) is used in most parts of Computer Vision. The most media grabbing case is:

  1. **Finding patterns in data**

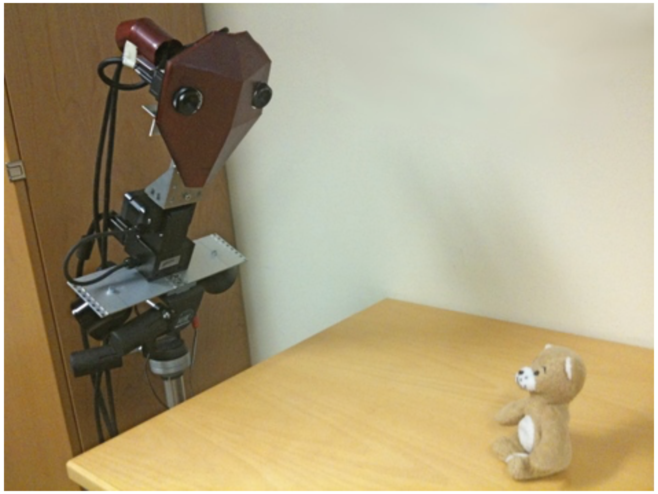
- Covered in e.g. TSBB19, TBMI26

Finding patterns in data

- ML used in recognition and matching (LE 8) in the form of learned features.



- Other applications include: object recognition, object tracking, image captioning etc.
  These are covered in TSBB19: Machine Learning for Computer Vision, HT1



Why machine learning?

- Machine Learning(ML) is used in most parts of Computer Vision. The most media grabbing case is:

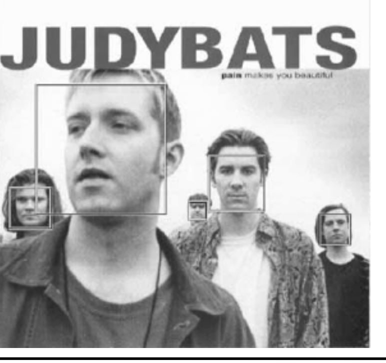
1. **Finding patterns in data**

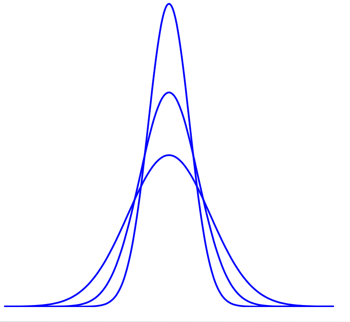
- Today we will look at two other use cases:

2. **Parameter tuning**
3. **Adaptation to changing conditions**

Parameter tuning

- Most Computer Vision systems are complex pieces of software.
- The more complex a system is, the more parameters it has. E.g. filter sizes, thresholds for detection etc. These need to be tuned!





Parameter tuning

- Parameter tuning, **the hacker way**:

  Try different values of the parameters, and look at the result on a test example.

Parameter tuning

- Parameter tuning, **the hacker way**:

  Try different values of the parameters, and look at the result on a test example.

- Results in **overfitting**: result is good on the test example, but bad in other cases.

Parameter tuning

- Parameter tuning, **the engineering way**:

  1. Collect a set of examples of the desired behaviour of an algorithm.
  2. Look for the parameters that produce the desired behaviour on the examples.

Parameter tuning

- Parameter tuning, **the engineering way**:

  1. Collect a set of examples of the desired behaviour of an algorithm.
  2. Look for the parameters that produce the desired behaviour on the examples.

- Be careful: do not tune too much! (overfitting again)

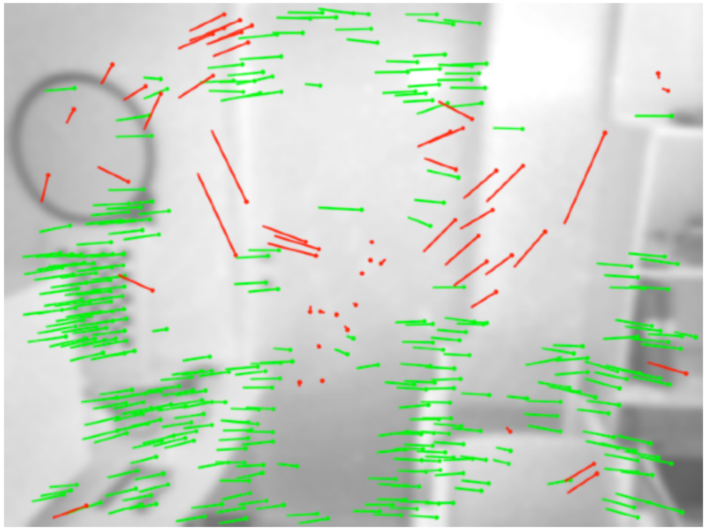
Parameter tuning

- Parameter tuning as **supervised learning**:

  1. Collect a training set
  2. Iteratively change the parameters to improve performance on the training set.
  3. Decide when to stop, by monitoring performance on a validation set.

  Parameters are found by minimizing a loss function that defines the desired behaviour.

Parameter tuning



- Example: KLT Tuning
  Rotate the camera, and fit a homography to tracks. This way you can automatically decide which motion vectors are good ($\mathbf{v} \in G$) and which are bad ($\mathbf{v} \in B$).
- Look for **tracker parameters** that minimise a **loss**.
  E.g.: $J(p_1,\ldots,p_N) = |B|/(|G|+|B|)$

Parameter tuning

- Best practice in parameter tuning is to use **supervised learning**:
- **Training set**
  - change parameters to minimize a **loss** on this.
- **Test set** (examples **not used** in learning/tuning)
  - tests generalization to new situations.
- **Validation set** (part of the training set)
  - used to decide when to stop tuning, and avoid **overfitting**.

Adaptation

- Computer Vision systems that are deployed in live situations face changing conditions. E.g. different illumination at night and during the day.

Adaptation

- Computer Vision systems that are deployed in live situations face changing conditions. E.g. different illumination at night and during the day.
- A convenient way to cope with changes, is to make the vision system adaptive. (an alternative is *invariance*, see LE8).

Adaptation

- Computer Vision systems that are deployed in live situations face changing conditions. E.g. different illumination at night and during the day.
- A convenient way to cope with changes, is to make the vision system adaptive.
- Example: Background models introduced later in this lecture.

Learning in Vision Systems

- Batch learning: *learn once, use forever*

- Online learning: *learn continuously*

Learning in Vision Systems

- Batch learning: *learn once, use forever*
  Is used to automatically tune parameters, features, classifiers etc.
- Online learning: *learn continuously*
  Is used to automatically adapt e.g. classifiers and trackers to changing conditions.

Learning paradigms

- Different learning situations/paradigms:

  Supervised learning
  Reinforcement learning
  Unsupervised learning

- Covered in depth in:
  TBMI26 Neural Networks and Learning Systems

Learning paradigms

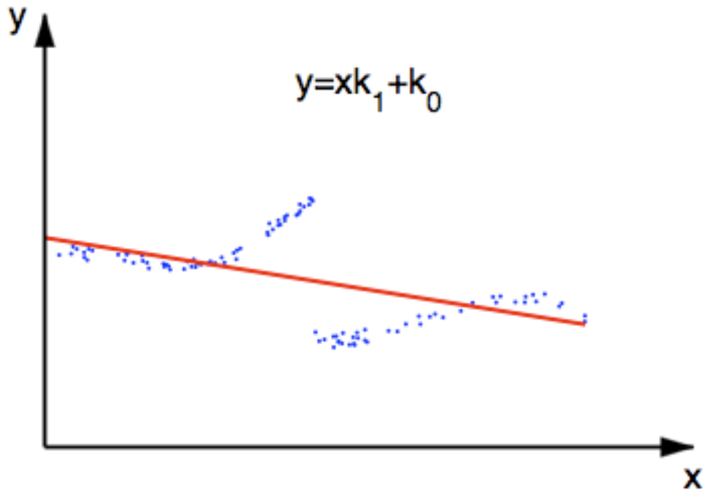
- Different learning situations/paradigms:

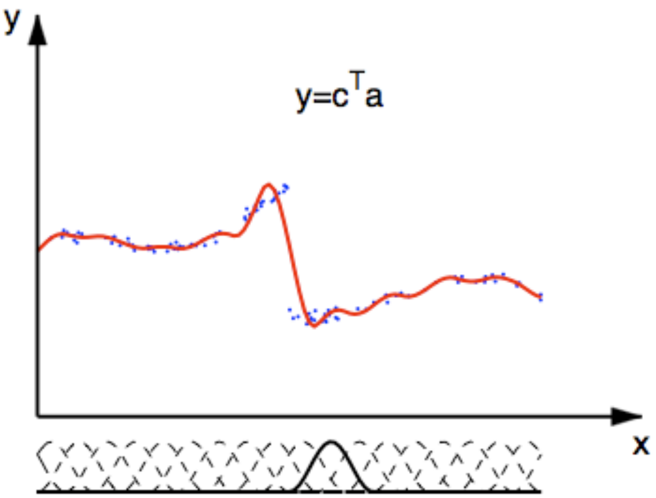
  Supervised learning
  Reinforcement learning
  Unsupervised learning ←this lecture

- Covered in depth in:
  TBMI26 Neural Networks and Learning Systems

Learning paradigms

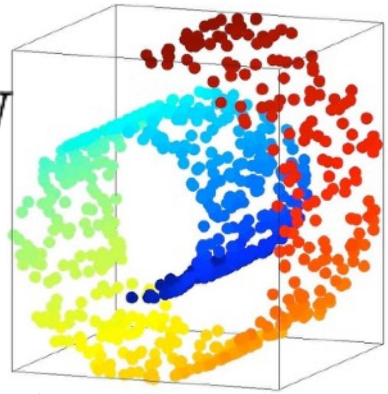
- **Supervised learning**
  learn **y**=f(**x**) from examples $\{\mathbf{x}_n, \mathbf{y}_n\}_1^N$
  = function approximation



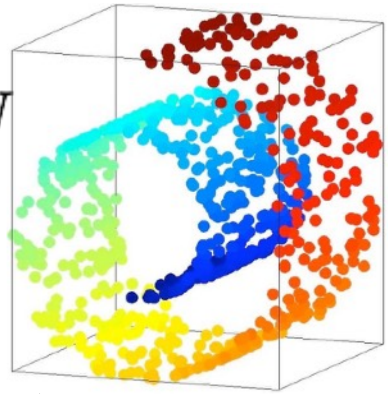


Learning paradigms

- **Unsupervised learning**
  learn **y**=f(**x**) from examples $\{\mathbf{x}_n\}_1^N$
  =*manifold learning* or *clustering*
  - Manifold learning finds low dimensional representations of high dimensional data. E.g. coordinates on a surface in nD.



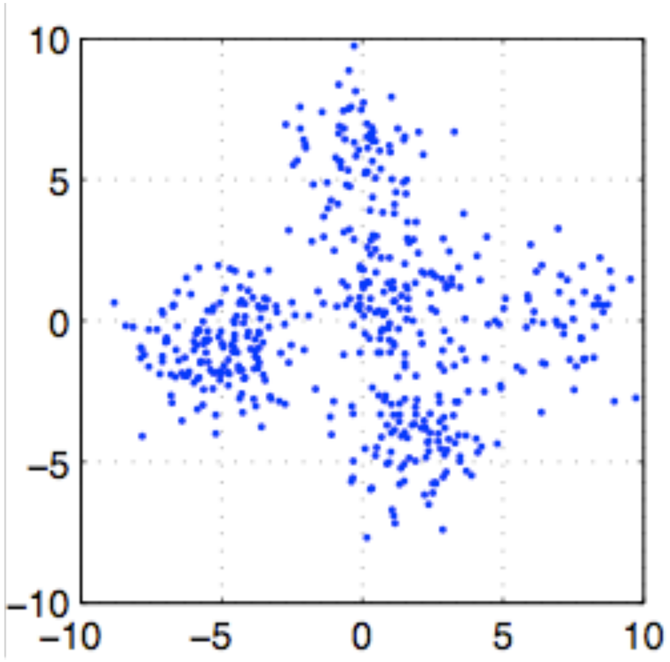
Learning paradigms



- **Unsupervised learning**
  learn **y**=f(**x**) from examples $\{\mathbf{x}_n\}_1^N$
  =*manifold learning* or *clustering*
  - Manifold learning finds low dimensional representations of high dimensional data. E.g. coordinates on a surface in nD.
- This lecture is mainly about **clustering**:
  $y \in \mathbb{N}$, i.e. each sample $\mathbf{x}_n$ is assigned a cluster *label*.

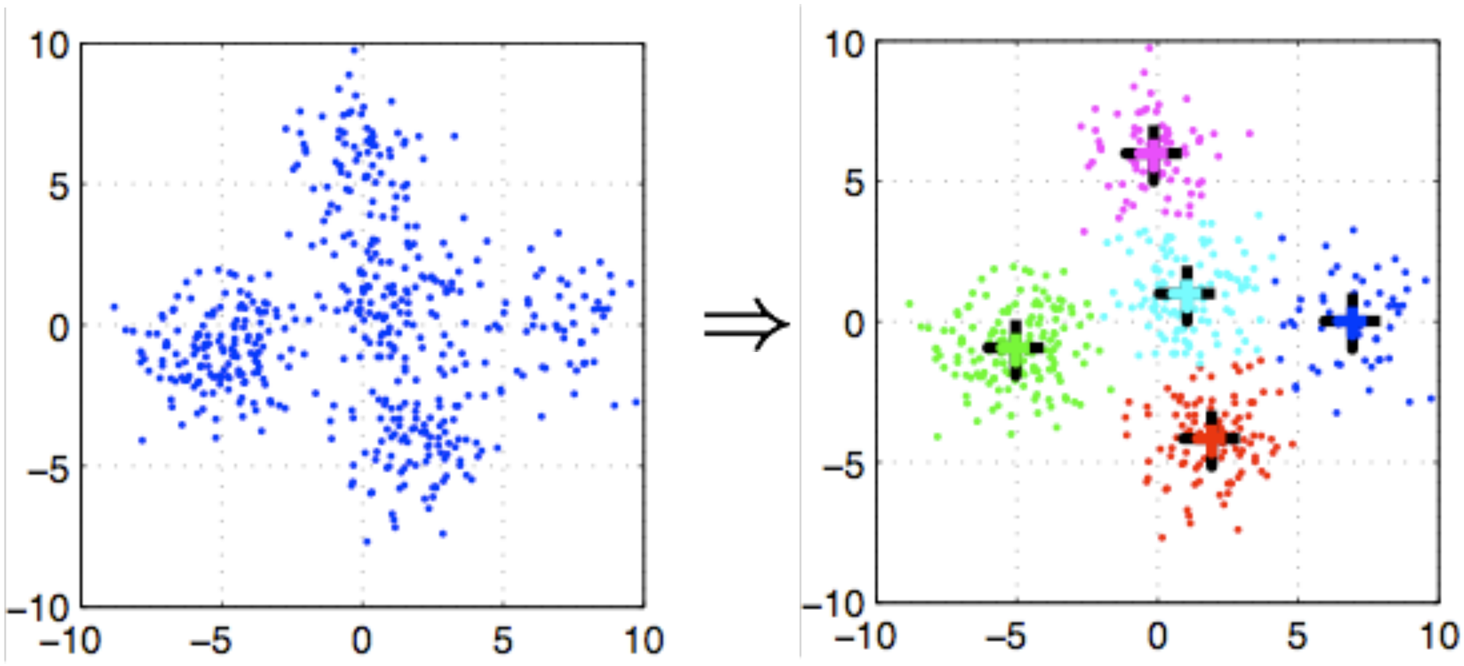
Clustering

– Our input is a set of data points $\{\mathbf{x}_n\}_1^N$



Clustering

– Each data point $\{\mathbf{x}_n\}_1^N$ is assigned a cluster label $y \in [1 \ldots K]$, and a prototype $\{\mathbf{p}_k\}_1^K$

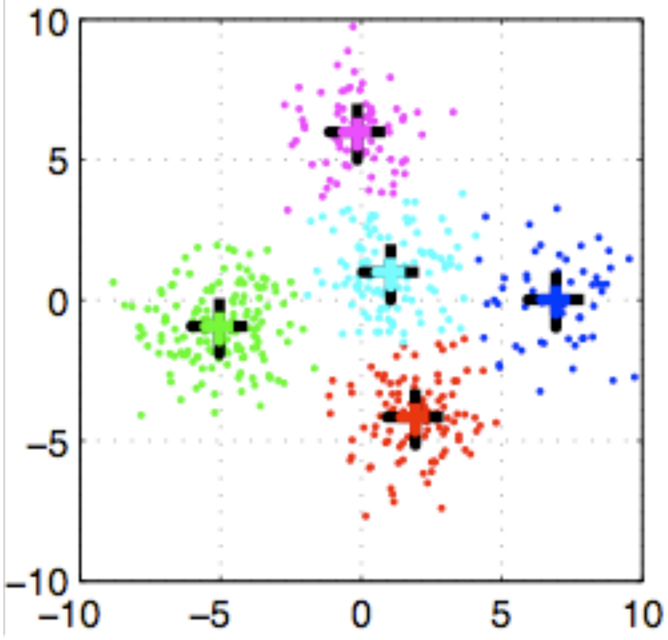


Clustering

- Each data point $\{\mathbf{x}_n\}_1^N$ is assigned a cluster label $y \in [1 \ldots K]$, and a prototype $\{\mathbf{p}_k\}_1^K$
- labels and prototypes are **latent** (hidden) variables that we want to **estimate**.
- Many algorithms, with different representations of the prototypes/clusters. We will now look at the **K-means algorithm**, and later **Expectation Maximisation** (EM)…

K-means clustering

- A good clustering has small distances between prototypes and samples within that cluster.



K-means clustering

- A good clustering has small distances between prototypes and samples within that cluster. Encoded in loss function:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

K-means clustering

- A good clustering has small distances between prototypes and samples within that cluster. Encoded in loss function:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

K-means clustering

- A good clustering has small distances between prototypes and samples within that cluster. Encoded in loss function:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

K-means clustering

- A good clustering has small distances between prototypes and samples within that cluster. Encoded in loss function:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

K-means clustering

- A good clustering has small distances between prototypes and samples within that cluster. Encoded in loss function:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

K-means clustering

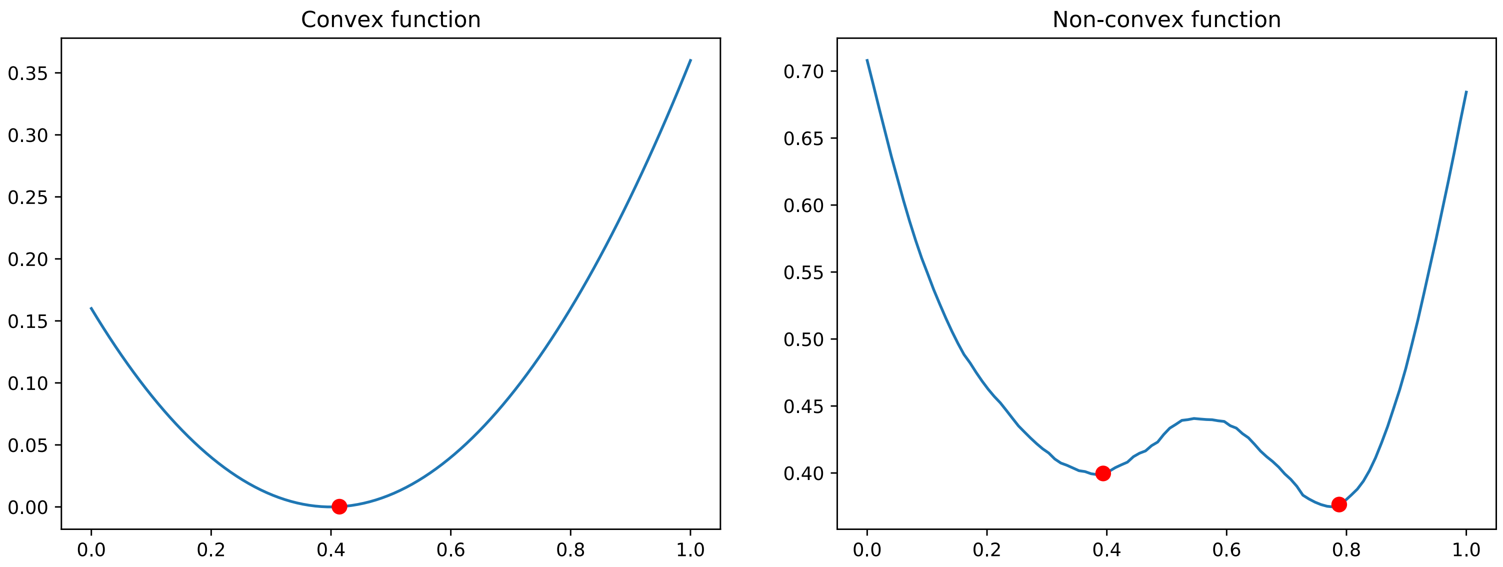
- Loss function:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

- **Non-convex problem**. What is this?
- K-means clustering [MacQueen'67] is a useful **heuristic**, that iteratively improves on an initial clustering.

Convexity

- Convex implies one local min (global min)



K-means clustering

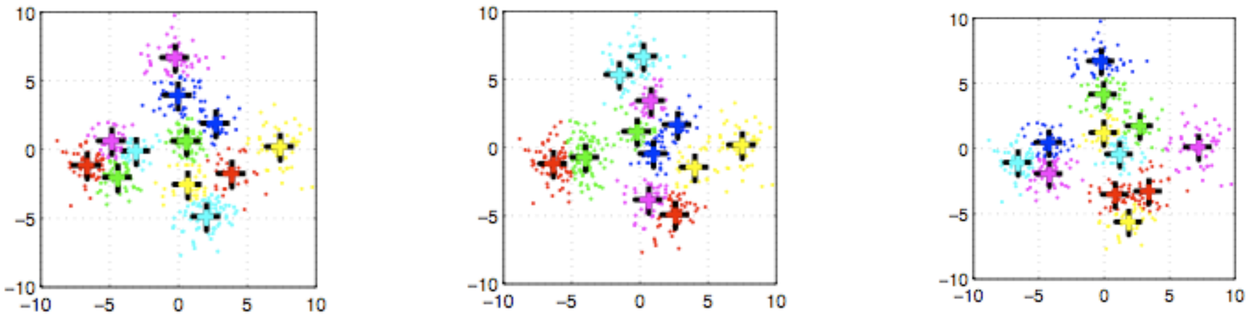
1. Pick random sample points as cluster prototypes.
2. Assign cluster labels $\{y_n\}_1^N$ to samples $\{\mathbf{x}_n\}_1^N$ according to prototype distances $d_k^2 = ||\mathbf{x}_n - \mathbf{p}_k||^2$
3. Assign prototypes as averages of samples within cluster: $$\mathbf{p}_k = \frac{1}{|\{y_n = k\}|} \sum_{n=1}^{N} \delta[y_n = k]\mathbf{x}_n$$
4. Repeat 2-3 until labels stop changing.

K-means issues

- K-means finds a *local min* of the loss:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

- Issue 1:Bad repeatability:



- Issue 2:What is the value of K?

K-means fixes

- Partial fixes for the local min problem:
  - Run the algorithm many times, and pick the solution with the lowest *J*.
  - choose a smarter initial value
  - The discrete label assignment makes the loss non-smooth. A smoother loss would mean fewer local minima.
- Steps 2,3 can be seen as special cases of the EM-algorithm [Dempster et al. 77] and EM has a smoother loss…
- To understand EM we first need to introduce *mixture models*.

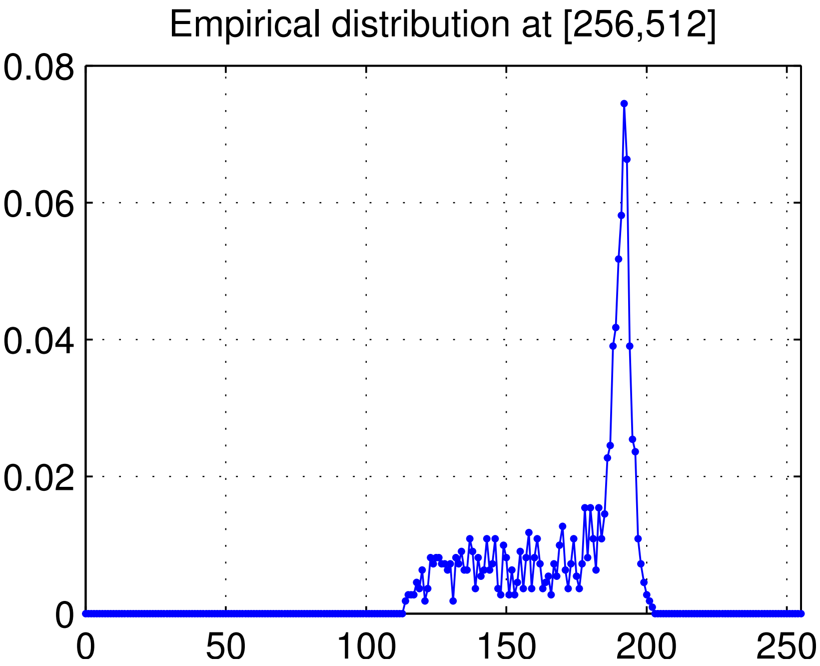
Mixture models

- A *generative model* for data that may come from several distributions.
- E.g. value of a particular pixel in a stationary camera:

  - shadow/no shadow
  - cloudy/sunny
  - temporary occlusion (flag or branches)

Mixture models

- Value of a particular pixel in a stationary camera: $p(I_{256,512})$



Mixture models

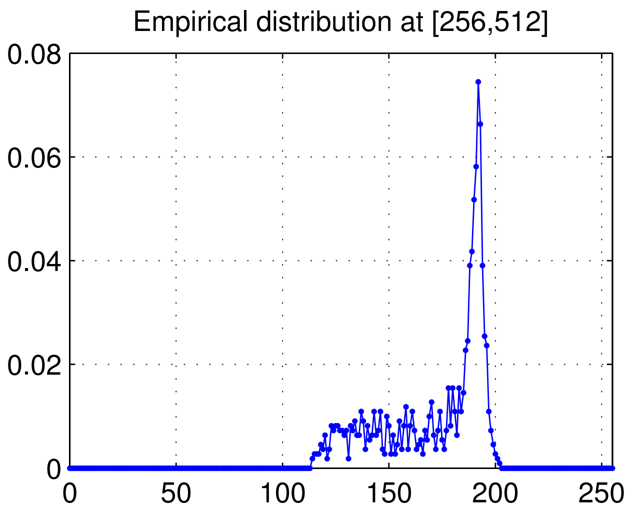
- We model the probability density of pixel intensity *I* as:

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

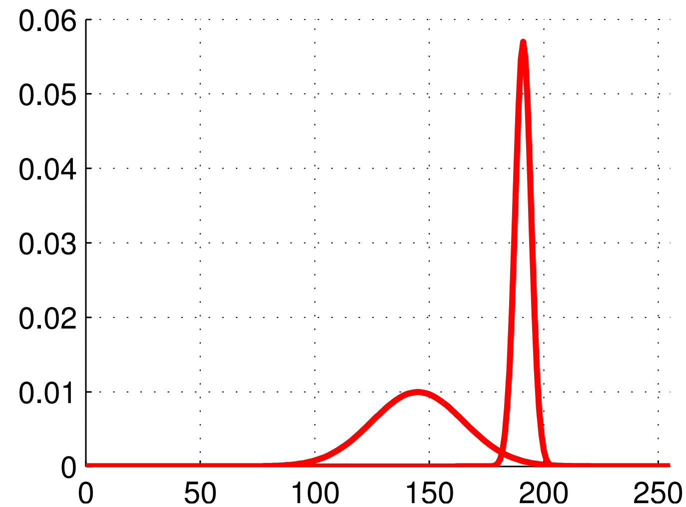
Mixture models

- We model the probability density of pixel intensity *I* as:

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$



$\approx$



Mixture models

- We model the probability density of pixel intensity *I* as:

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k) P(\Gamma_k)$$

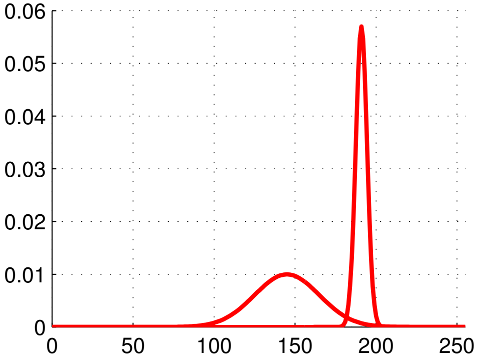
- *Mixture components*:

$$p(I|\Gamma_k)$$

e.g.

$$p(I|\Gamma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-0.5(I-\mu_k)^2/\sigma_k^2}$$

Gaussian mixture model (GMM)



Mixture models

- We model the probability density of pixel intensity *I* as:

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k) P(\Gamma_k)$$

- *Mixture probabilities*:

$$\sum_{k=1}^{K} P(\Gamma_k) = 1$$

Probability of being in a particular component.

Mixture models

- Gaussian mixture components*:*

$$p(I|\Gamma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-0.5(I-\mu_k)^2/\sigma_k^2}$$

- Notation conditioned on the parameters:

$$p(I|\mu_k,\sigma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-0.5(I-\mu_k)^2/\sigma_k^2}$$

- Also the mixture probabilities are parameters:

$$P(\Gamma_k) = \pi_k \ , \ \text{where} \ \sum_k \pi_k = 1$$

A generative model

- The mixture model is a **generative model**.
- This means that it can generate samples.

How?

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

A generative model

- The mixture model is a **generative model**.
- This means that it can generate samples.
  How?

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

- A: First draw component (How?), then draw sample from that component's distribution.

Expectation Maximisation

- Given a set of measurements, $\{I_n\}_1^N$ how do we estimate the parameters of the mixture distribution *p(I)*?

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

Expectation Maximisation

- Given a set of measurements, $\{I_n\}_1^N$ how do we estimate the parameters of the mixture distribution *p(I)*?

$$p(I|\{\pi_k, \mu_k, \sigma_k\}_1^K) = \sum_{k=1}^{K} \pi_k p(I|\mu_k, \sigma_k)$$

- This can be done with the EM algorithm.
- Note similarities with K-means below.

Expectation Maximisation

- Maximize a loss which is the log likelihood of all samples:

$$J(\Theta) = \log \left(\prod_{n=1}^{N} p(I_n|\Theta) \right) = \sum_{n=1}^{N} \log p(I_n|\Theta)$$

Expectation Maximisation

- Maximize a loss which is the log likelihood of all samples:

$$J(\Theta) = \log\left(\prod_{n=1}^{N} p(I_n|\Theta)\right) = \sum_{n=1}^{N} \log p(I_n|\Theta)$$

- Here Θ, is a vector that includes parameters of the mixture and component *assignments* (cf. labels in K-means):

$$\Theta = (\pi_1, \ldots, \pi_K, \sigma_1, \ldots, \sigma_K, \mu_1, \ldots, \mu_K, a_{11}, \ldots, a_{KN})$$

Expectation Maximisation

- Maximize a loss which is the log likelihood of all samples:

$$J(\Theta) = \sum_{n=1}^{N} \log p(I_n|\Theta)$$

- To do this we alternate between:
  E: compute assignments, from sample likelihoods using current model, Θ_{t-1}
  M: estimate other model parameters in Θ_t, given the assignments

Expectation Maximisation

- The E-step for a mixture:

$$p(I|\{\pi_k,\mu_k,\sigma_k\}_1^K) = \sum_{k=1}^{K} \pi_k p(I|\mu_k,\sigma_k)$$

- Computes the *assignments* (aka. responsibilities) according to:

$$\tilde{a}_{kn} = \pi_k p(I_n|\mu_k,\sigma_k)$$

$$a_{kn} = \tilde{a}_{kn} / \sum_{l=1}^{K} \tilde{a}_{ln}$$

Expectation Maximisation

- The M-step updates the mixture probabilities:

$$\pi_k = P(\Gamma_k) = \frac{1}{N}\sum_{n=1}^{N} a_{kn}$$

- and mixture parameters (assuming a GMM):

$$\mu_k = \frac{1}{\sum_{n=1}^{N} a_{kn}} \sum_{n=1}^{N} a_{kn} I_n$$

$$\sigma_k^2 = \frac{1}{\sum_{n=1}^{N} a_{kn}} \sum_{n=1}^{N} a_{kn} (I_n - \mu_k)^2$$

The EM Algorithm

1. Postulate a mixture distribution.

2. **E**: Compute assignments, a_{kn}, for samples $\{I_n\}_1^N$, using the current mixture model.

3. **M**: Use assignments to update mixture model parameters.

4. Repeat 2-3 until convergence.

Expectation Maximisation

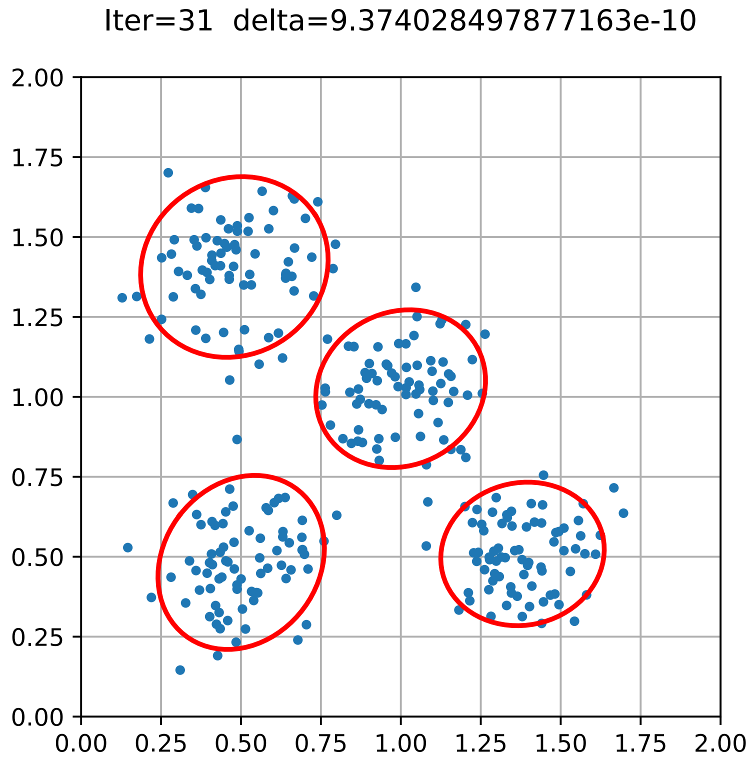
- Generalizes to higher dimensions.
- e.g. in 2D we have 5 parameters in each mixture component:

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \quad \mathbf{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix}$$

- Just like K-means,
  EM also finds a local min.

Expectation Maximisation

- Demo for 2D case:

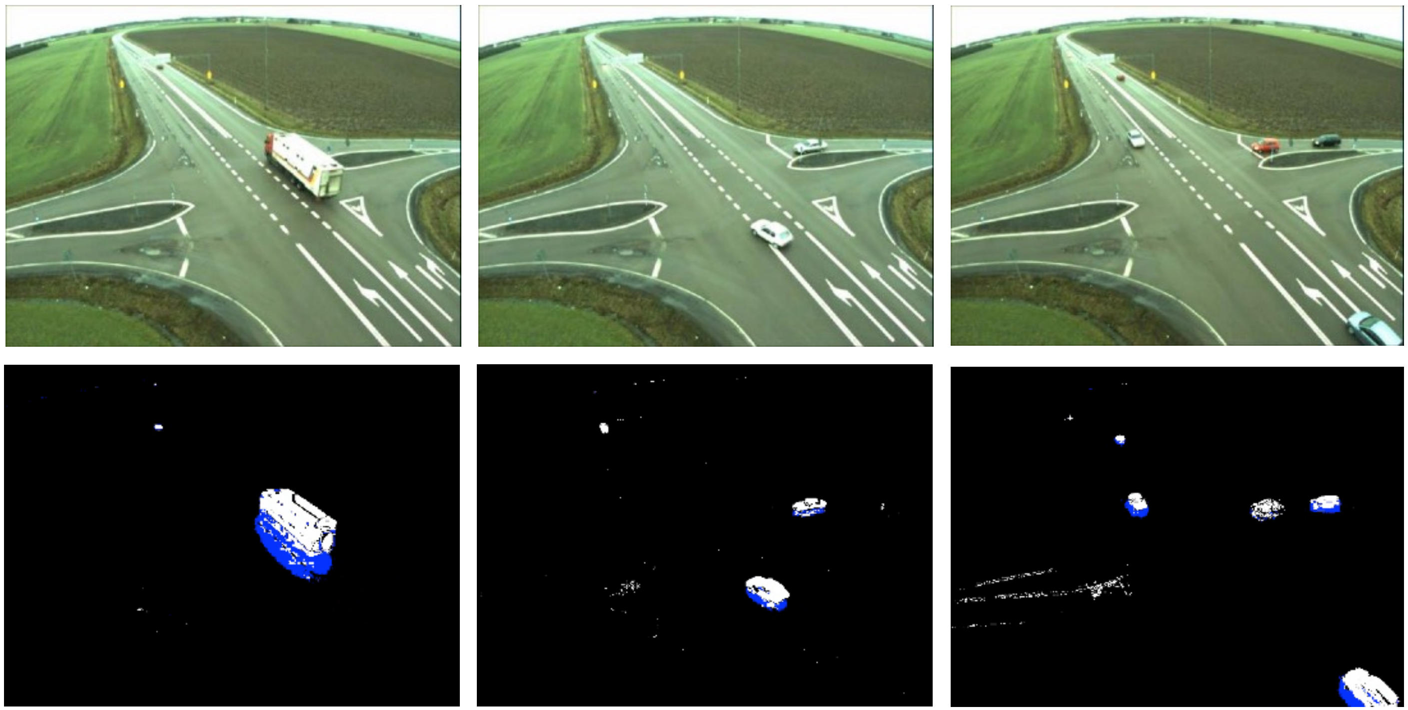


Background modelling

- A popular application of mixture models is **background modelling** (SHB 16.5.1):
  - Estimate a mixture model for the image *in each pixel.*
  - Pixel values far from the mixture are seen as foreground pixels.
  - Popular way track e.g. people and cars in **stationary** surveillance cameras.
  - Fast compared to motion estimation.

Background modelling

- Background modelling+shadow detection



- CVL Master thesis of John Wood 2007

Background modelling

- Samples now arrive one at a time.
- EM uses a batch update:

$$\mu_k = \frac{1}{\sum_{n=1}^{N} a_{kn}} \sum_{n=1}^{N} a_{kn} I_n$$

$$\sigma_k^2 = \frac{1}{\sum_{n=1}^{N} a_{kn}} \sum_{n=1}^{N} a_{kn} (I_n - \mu_k)^2$$

- On-line update is needed

Background modelling

- Samples now arrive one at a time.
- On-line update:

$$\mu_k[n] = (1-\alpha)\mu_k[n-1] + \alpha I_n$$

$$\sigma_k^2[n] = (1-\alpha)\sigma_k^2[n-1] + \alpha(I_n - \mu_k[n-1])^2$$

$$\pi_k[n] = (1-\alpha)\pi_k[n-1] + \alpha a_{kn}$$

- How to design $\alpha(a_{kn}, \pi_k, k)$ can be investigated in project 1.

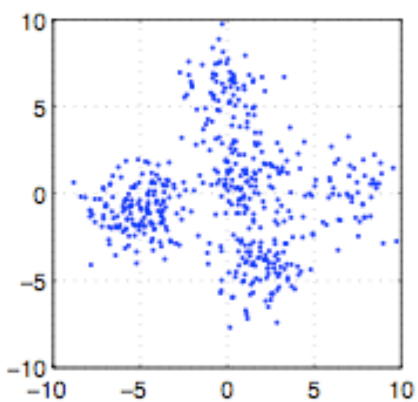
Mean-shift Clustering

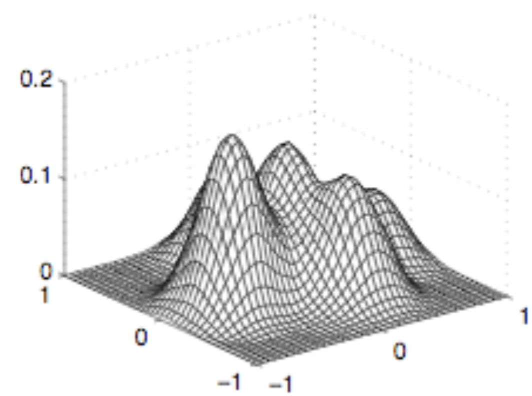
- A proper solution to the local min problem is to find *all* local minima.
- Two steps:
  - Mean-shift filter (mode seeking)
  - Clustering

Kernel density estimate

- For a set of sample points $\{\mathbf{x}_n\}_1^N$ we define a continuous PDF-estimate as:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)$$





Kernel density estimate

- For a set of sample points $\{\mathbf{x}_n\}_1^N$ we define a continuous PDF-estimate as:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)$$

- K() is a kernel, e.g. $K(\mathbf{x}) = c\exp\left(-\mathbf{x}^T\mathbf{x}/2\right)$
- h is the kernel scale.

Mode seeking

- By *modes* of a PDF, we mean the local peaks of the kernel density estimate.
  - These can be found by gradient ascent, starting in each sample.
  - If we use the Epanechnikov kernel,

$$K_E(\mathbf{x}) = \begin{cases} c(1 - \mathbf{x}^T\mathbf{x}) & \text{if } \mathbf{x}^T\mathbf{x} \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

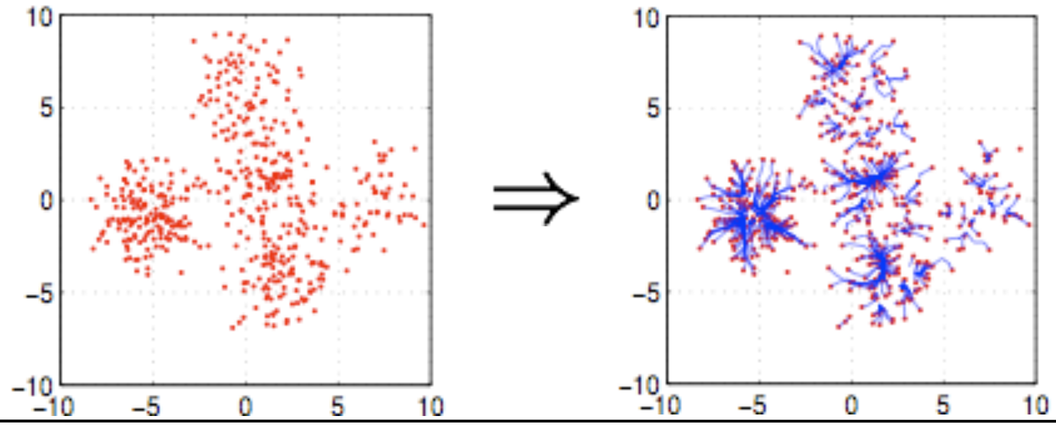
    a particularly simple gradient ascent is possible.

Mean-shift filtering

- Start in each data point, $\mathbf{m}_n = \mathbf{x}_n$
- Move to position of local average

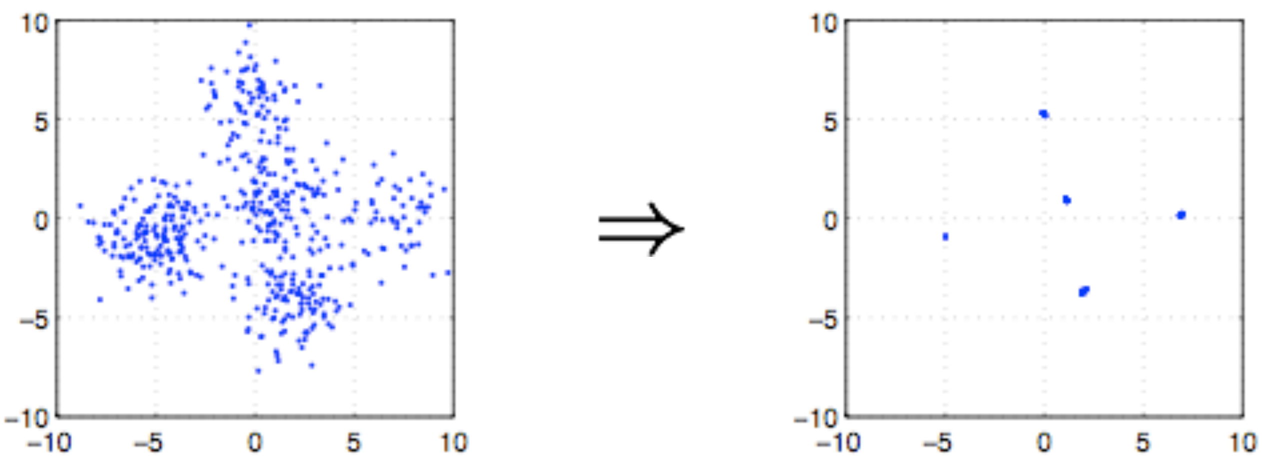
$$\mathbf{m}_n \leftarrow \text{mean}\left\{\mathbf{x}_n : \mathbf{x}_n \in S(\mathbf{m}_n)\right\}$$

- Repeat step 2 until convergence.



Mean-shift clustering

- After convergence of the mean-shift filter, all points within a certain distance (e.g. *h*) are said to constitute one cluster.

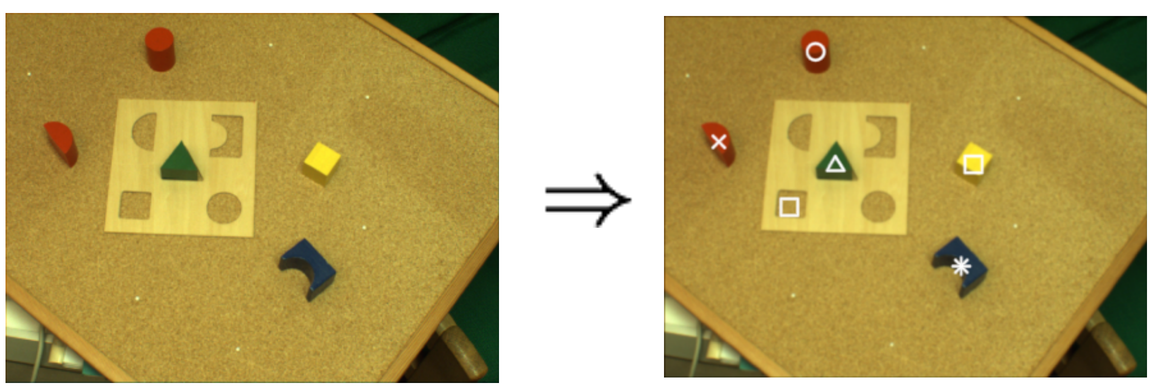


Pose estimation

– Mean-shift can be used for “continuous voting” in pose estimation.

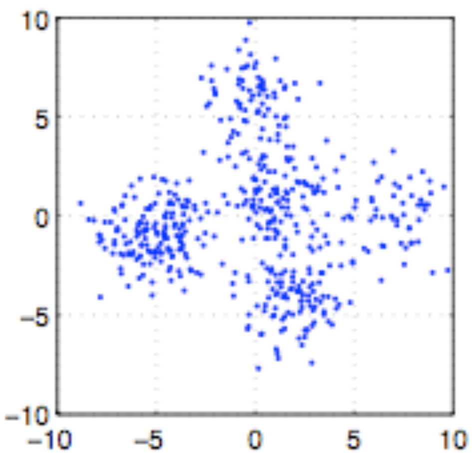
– Each local invariant feature (e.g. SIFT or MSER) will cast a vote (sample point)

$$\mathbf{x} = \begin{pmatrix} x_0 & y_0 & \alpha & s & \varphi & \theta & \mathtt{type} \end{pmatrix}^T$$

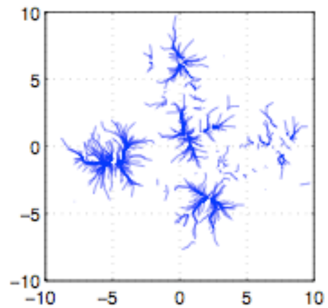


Mean-shift

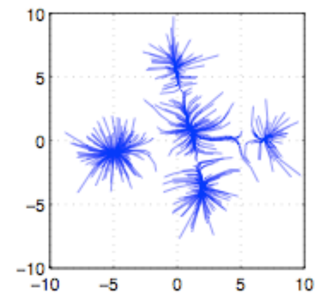
- Choice of kernel scale affects results



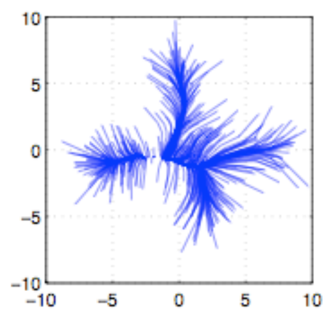
⇒



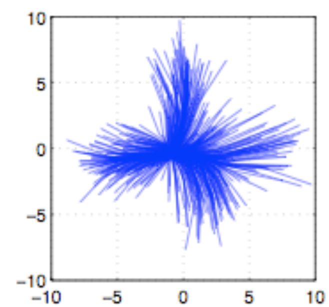
h=1.2 => 26 clusters



h=3.2 => 5 clusters



h=5.2 => 2 clusters



h=7.2 => 1 cluster

Mean-shift

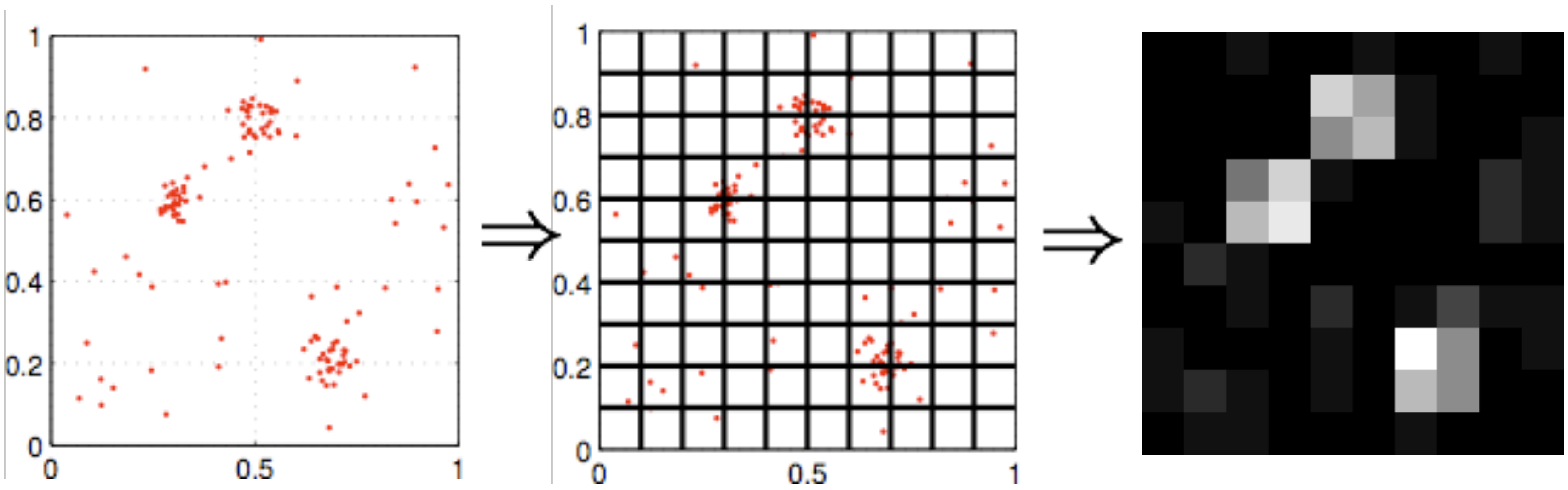
- For the Epanechnikov kernel, the algorithm is quite fast.
- The Gaussian kernel is another popular choice.
- There is also a scale adaptive version of meanshift, that works in a manner similar to EM in each iteration (slower).

Generalised Hough Transform

- Another way to find modes of a PDF is to quantize the parameter space into accumulator cells.
- Each sample then casts a vote in one or several cells.
- This is called the *Generalised Hough Transform* (GHT).

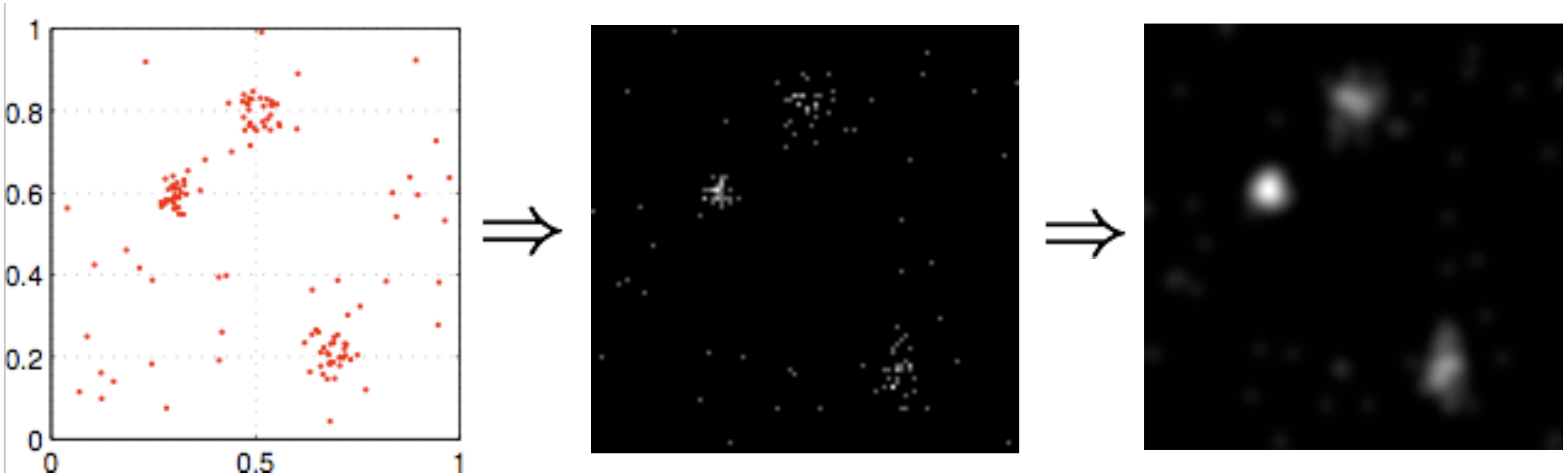
Generalised Hough Transform

- Non-iterative ⇒ constant time complexity.



Generalised Hough Transform

- Quantisation can be dealt with by increasing the number of cells, and blurring.

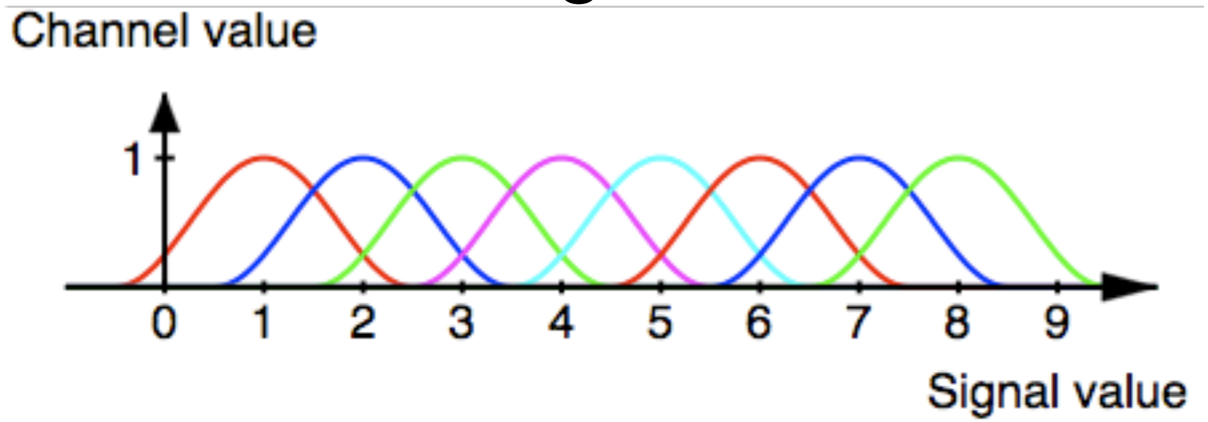


Channel Representation

- A similar technique is to use averaging in *channel representation*.
  - By first quantizing, and then blurring, we are actually introducing aliasing of the PDF.
  - Better to directly sample the kernel density estimate at regularly sampled positions.
  - Density of samples is regulated by the kernel scale.

Channel Representation

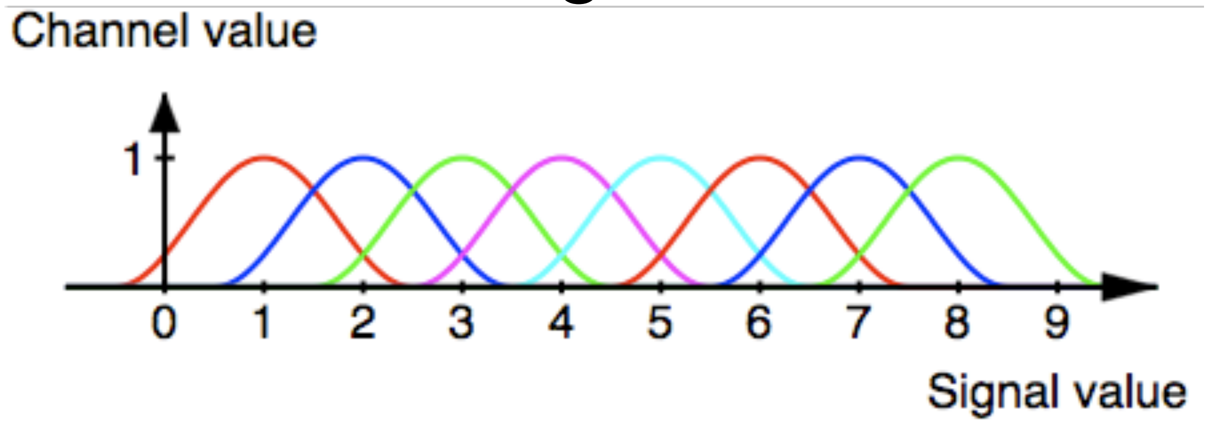
- Channel encoding



$$x = 4 \;\Rightarrow\; \mathrm{enc}(x) = \mathbf{x} = \begin{bmatrix} B(x-1) & B(x-2) & \ldots & B(x-8) \end{bmatrix}^T$$

Channel Representation

- Channel encoding



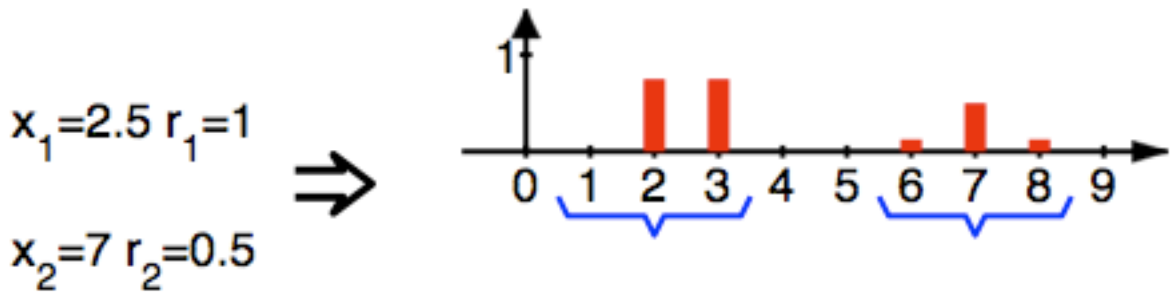
$$x = 4 \Rightarrow \mathrm{enc}(x) = \mathbf{x} = \begin{bmatrix} 0 & 0 & 0.25 & 1 & 0.25 & 0 & 0 & 0 \end{bmatrix}^T$$

- Channel decoding

$$\hat{x} = \mathrm{dec}(\mathbf{x})$$

Channel Representation

- A local decoding is necessary in order to decode a multi-valued channel representation.



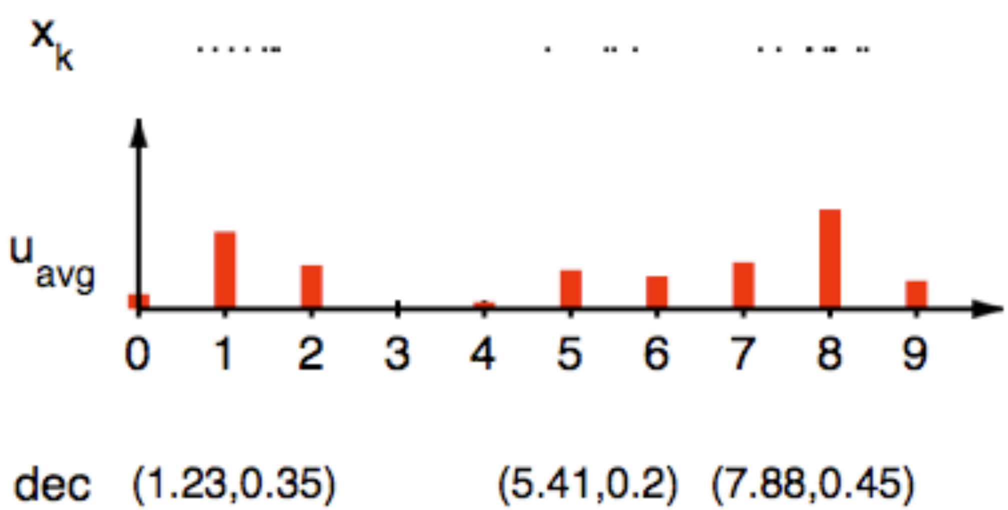
- That is

$$\hat{x}_1 = \mathrm{dec}(x_1 \ldots x_3) \qquad \hat{x}_2 = \mathrm{dec}(x_6 \ldots x_8)$$

- Decoding formula depends on the kernel.

Channel Clustering

- Channel encode data points, $\mathbf{x}_n = \text{enc}(x_n)$
- Average channel vectors $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$
- Compute all decodings $(\hat{x}, \hat{r})$



Channel Clustering

- The decoding step computes *location*, *density*, and *standard deviation* at mode.
- Optimal decoding is expensive, but fast heuristic decodings exist.
- It can be shown [Forssén 04] that averaging in channel representation is equivalent to a regular sampling of a kernel density estimator.

Summary

- This was a quick overview of clustering, and related techniques.
- The main purpose with learning is to make Computer Vision systems adapt to data.
- The alternative, to manually tune parameters, works for small static problems, but does not scale and cannot adapt to changes.

Course events this week

- Thursday (tomorrow): Lab1
  Material on the course web page.
  Extensive preparation is necessary to finish on time.

- Friday: Projects start
  Introductory lecture
  Assignments into groups (5/4 per group)
  If you cannot be there, let me know!